Reducing Coarse Grids Contention in a Parallel Algebraic Multigrid

Andrey Prokopenko

Jonathan Hu, Siva Rajamanickam

SIAM Parallel Processing
19 February 2014

Sandia National Laboratories

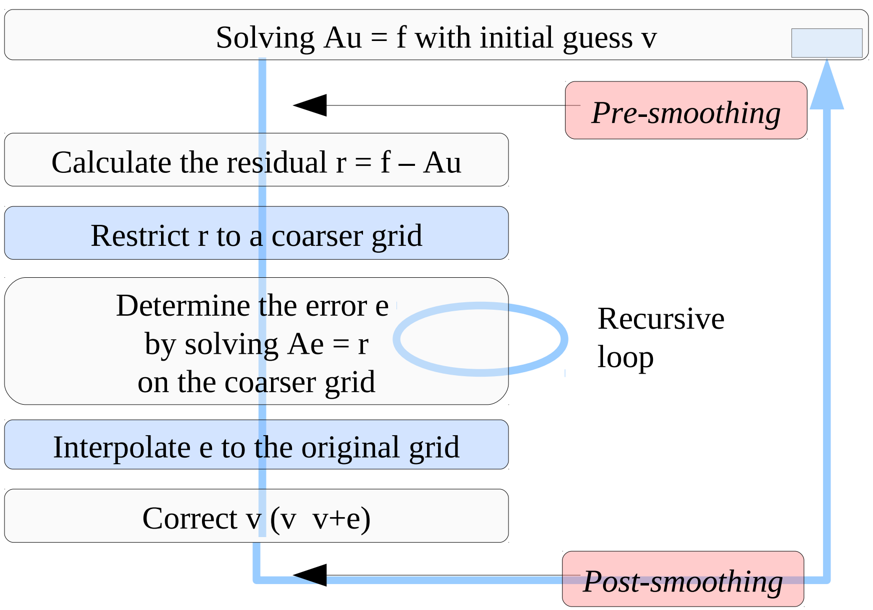
SAND 2014-1299C

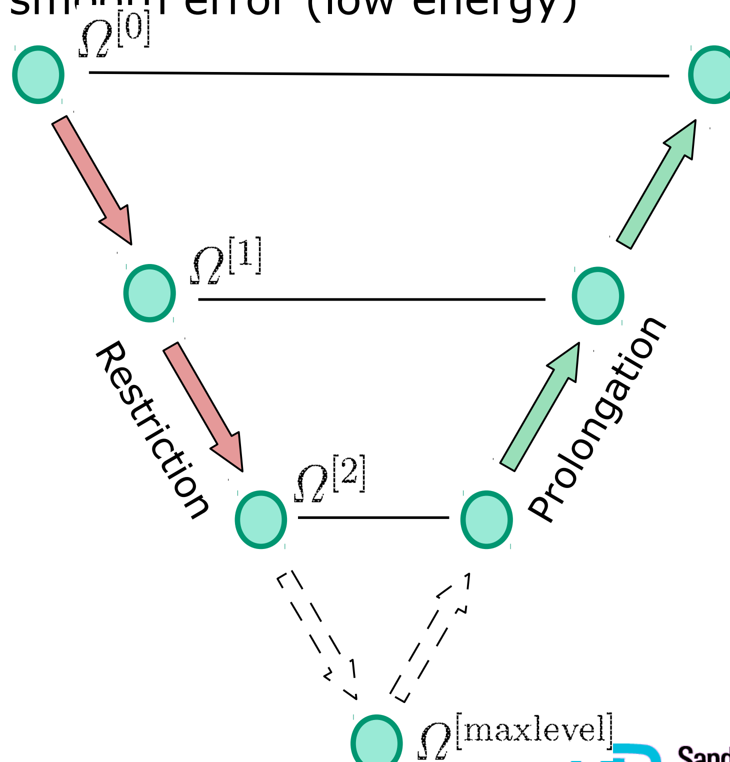
Sandia National Laboratories is a multi-program laboratory operated by Sandia Corp, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Sandia National Laboratories

Algebraic multigrid (AMG)

- Iterative method for solving linear equations
- Commonly used as a preconditioner
- Idea: capture error at multiple resolutions using grid transfer operator:
  - **Smoothing** damps the oscillatory error (high energy)
  - **Coarse grid correction** reduces the smooth error (low energy)





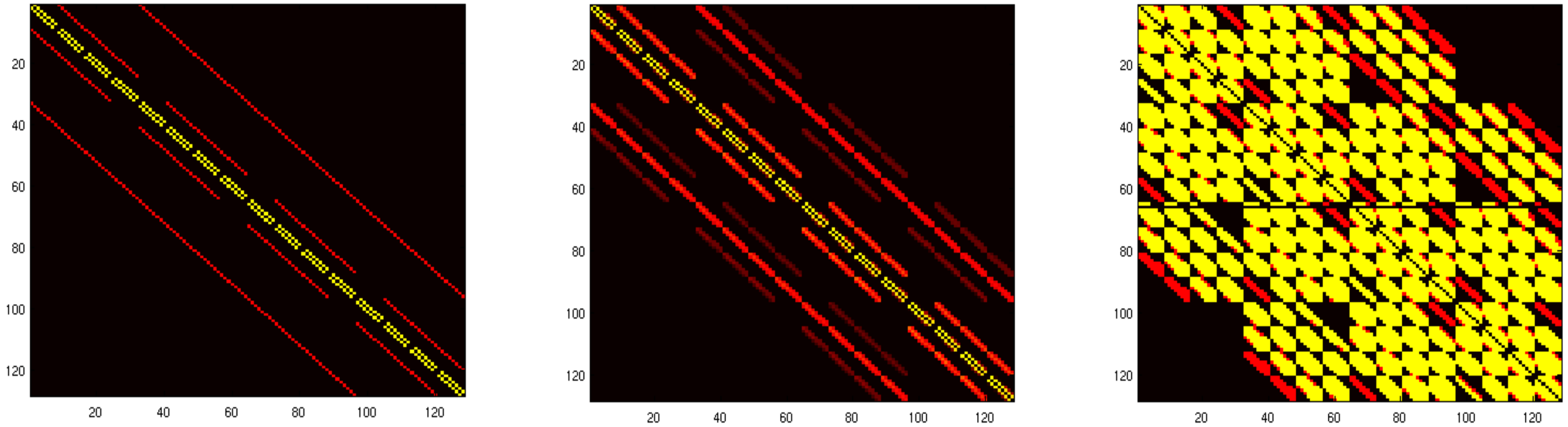
Increased communication

Coarse grids typically have increased communication

- Fewer flops per node
- Denser coarse matrices

  For instance, for elasticity number of nonzeros per row:

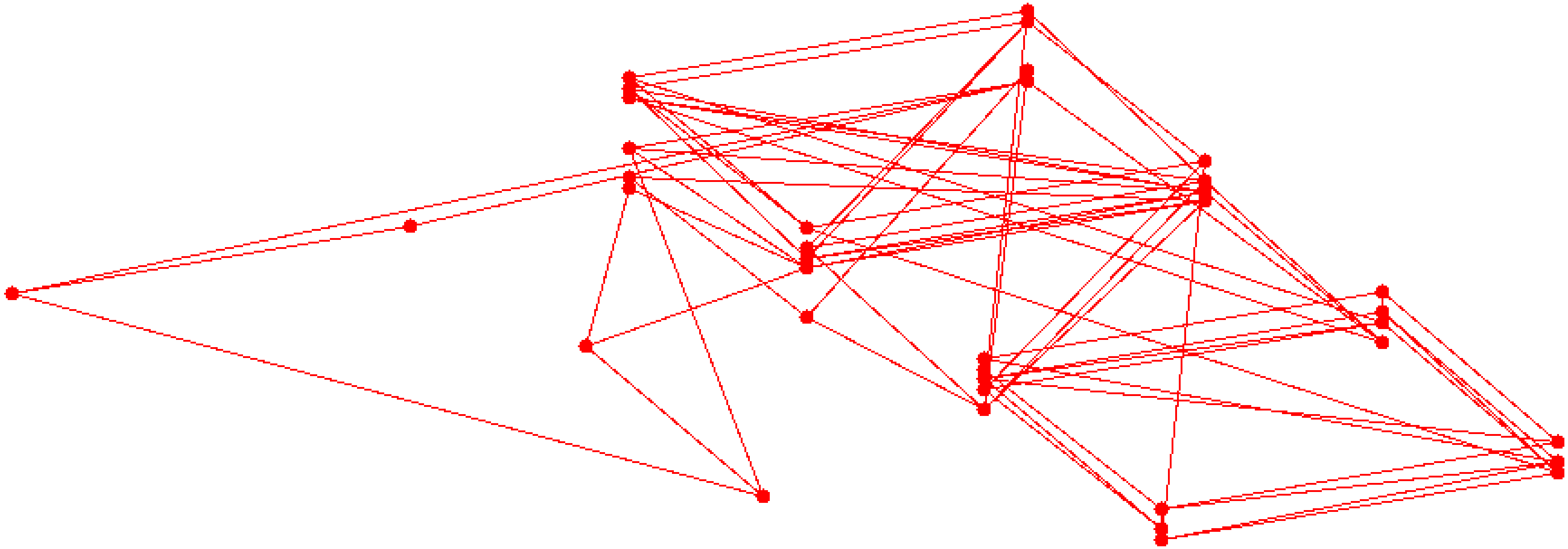
  81 → 206 → 373 → 691



Sparse allocations

Large clusters are used by many users who can submit thousands of jobs. Therefore, to reduce time in queues many schedulers can construct **sparse allocations**. These could be problematic:

- Messages can travel long distances between two processors;
- Some links can become oversaturated



Outline

- Motivation
- Repartitioning
- Mapping algorithms
  - Mapping for sparse allocations
  - Mapping reducing data migration
- Conclusions and future work

Other approaches

- *Non-Galerkin AMG*
  J. Schroder, R. Falgout

- *Additive AMG*
  P. Vassilevski, U. Yang

- *Coarse level data redundancy*
  H. Gahvari, W. Gropp, K. Jordan, M. Schultz, U. Yang

Why repartitioning?

Multiple reasons to do repartitioning:

- Mitigate increase in complexity of SA-AMG

  Uncoupled aggregation without repartitioning produces inner-boundary effects

- Improve load balancing

- Reduce communication

  Fewer parts => less communication

Repartitioning steps

The decision to repartition depends on several heuristics:

- Load imbalance
- Load per processor (i.e., number of DOFs/processor)

The repartitioning itself is done in several stages:

1. Compute new partitions
2. **Map partitions to remaining processors**
3. Do data redistribution

Repartitioning affects:

- Setup phase
  - Data redistribution
  - Coarser levels construction
- Solve phase
  - Matrix-vector multiplication

Minimizing data contention

Goal: improve solve time on sparse allocations

Given **data**:

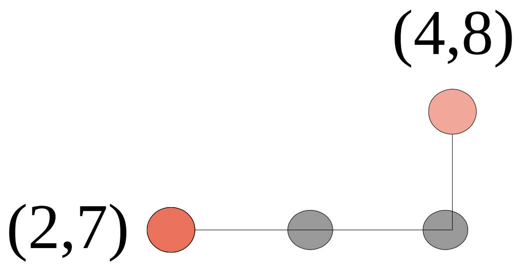
- Part coordinates (averaged)
- Application communication graph
- Processor (core) coordinates

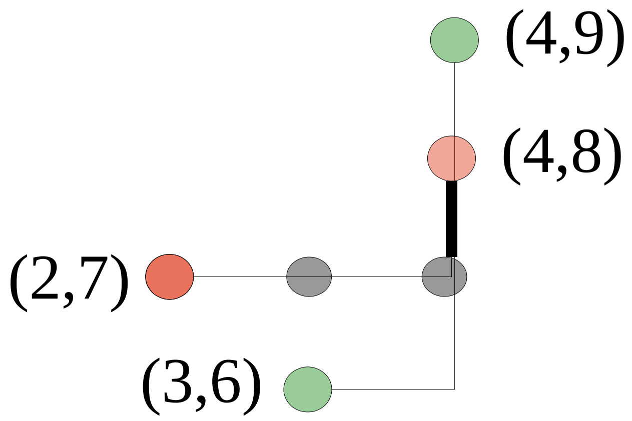
try to improve the following **metrics**:

- Average hop count
- Congestion

assuming that:

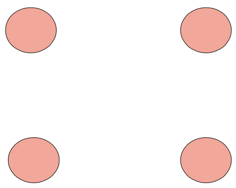
- Messages always take the shortest route
- Only static routing (no dynamic)



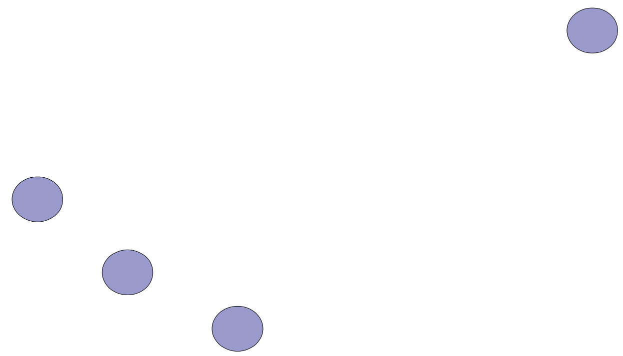


Minimizing data contention

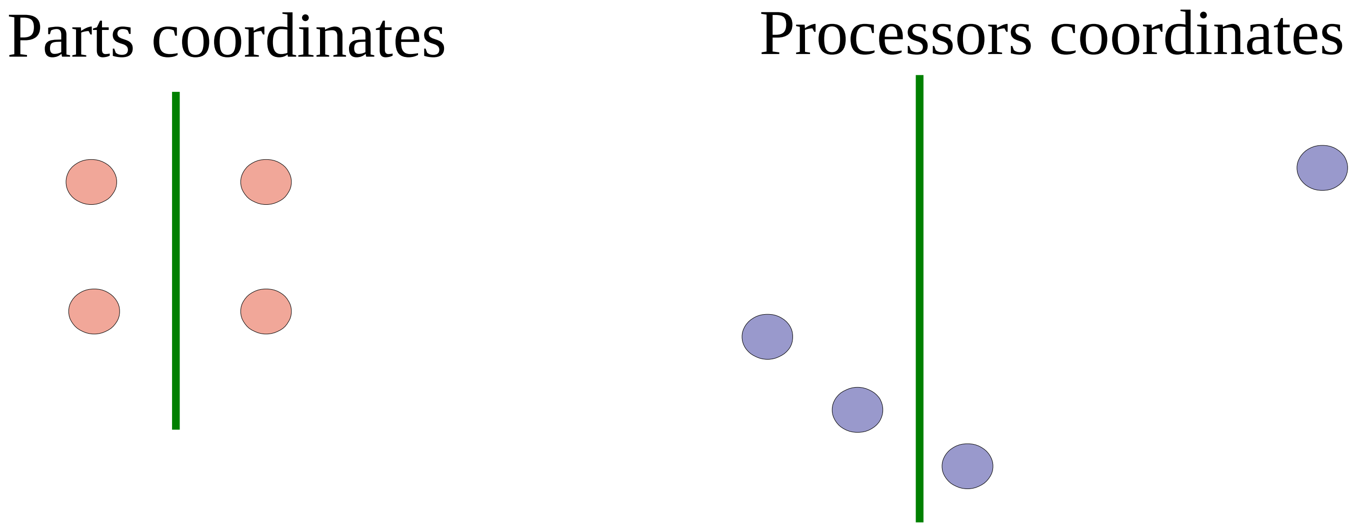
Parts coordinates



Processors coordinates

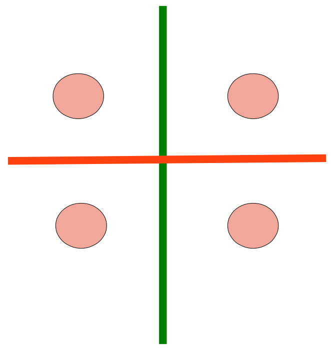


Minimizing data contention

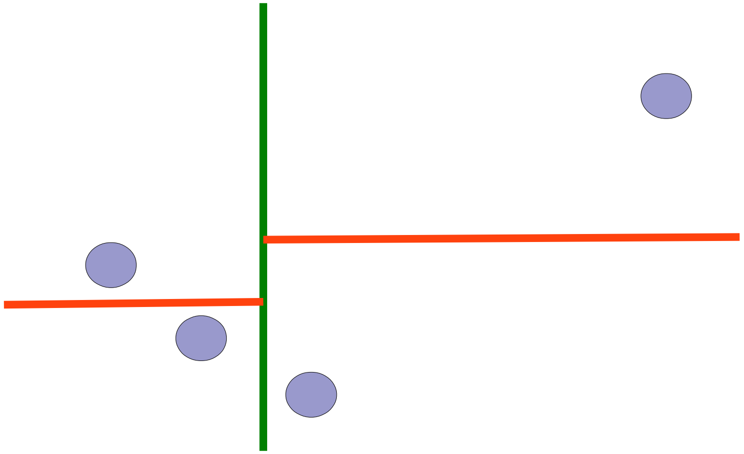


Minimizing data contention

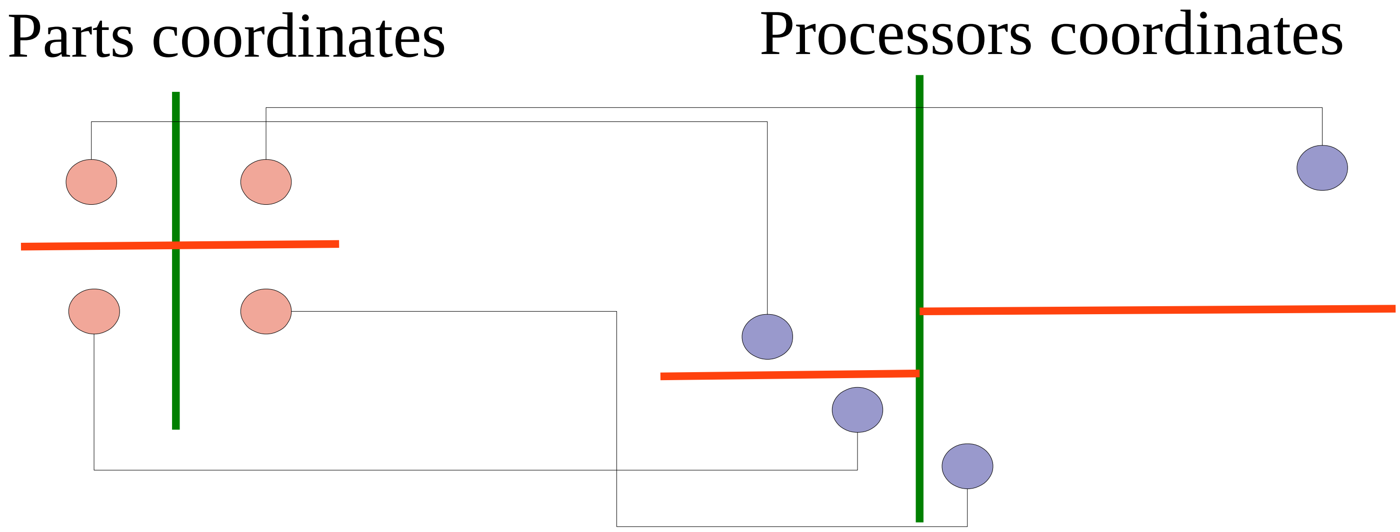
Parts coordinates



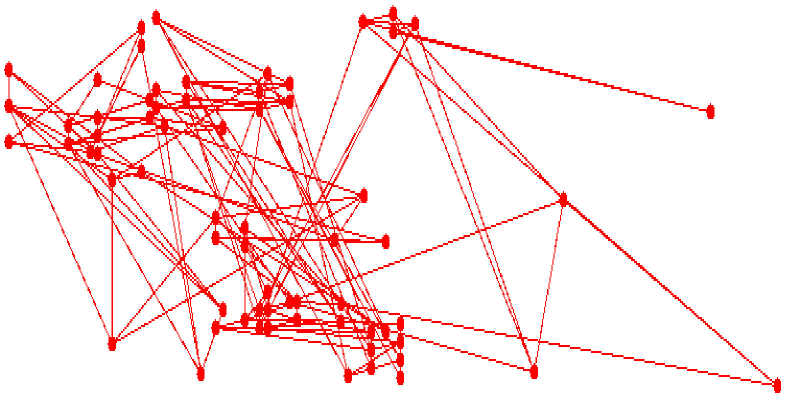
Processors coordinates



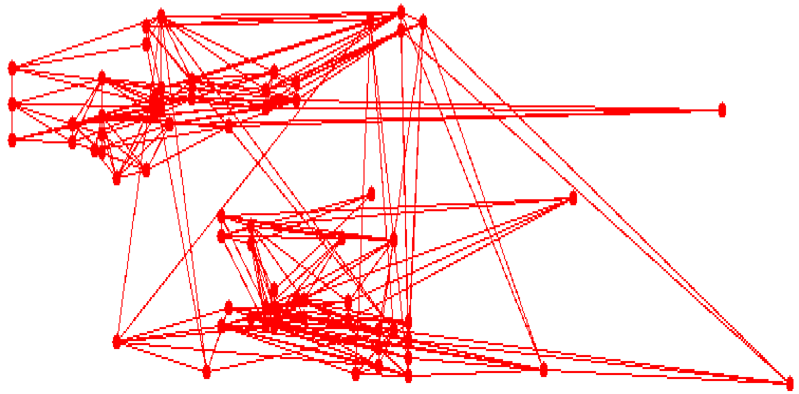
Minimizing data contention



Minimizing data contention



Avg hop count: 2.02
Congestion: 5.03



Avg hop count: 1.32
Congestion: 3.12

Bipartite graph matching

Goal: minimize setup time (and, possibly, solve time) for any allocation

Given **data**:

- Number of each part DOFs for each subdomain

try to improve the following **metrics**:

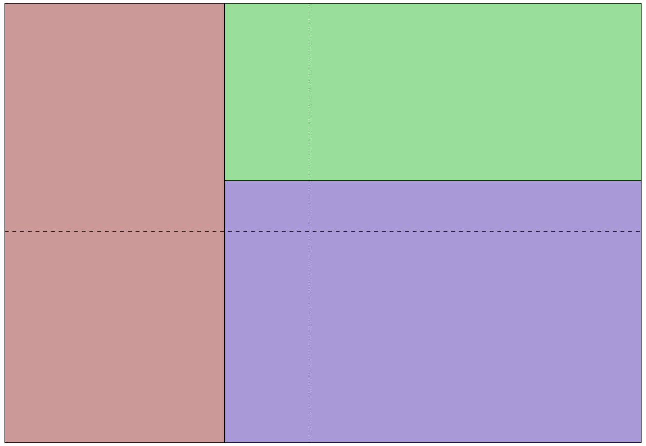
- Number of DOFs staying on the same processor

assuming that:

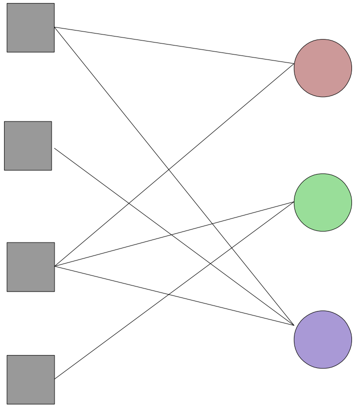
- Less redistribution leads to faster performance

Bipartite graph matching

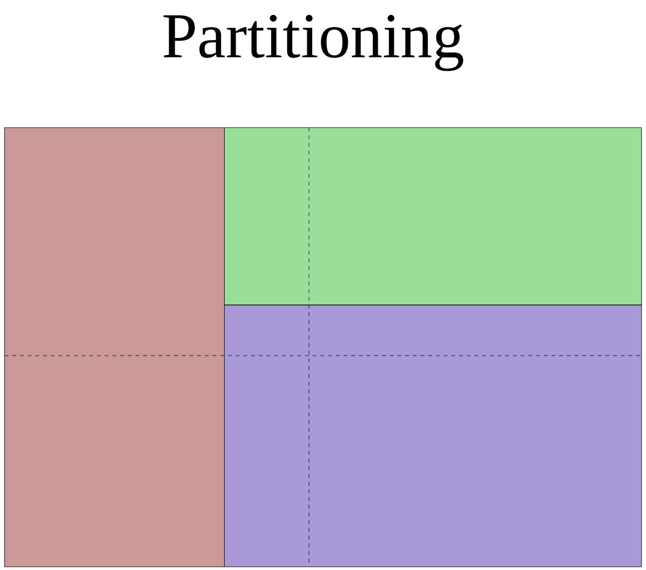
Partitioning

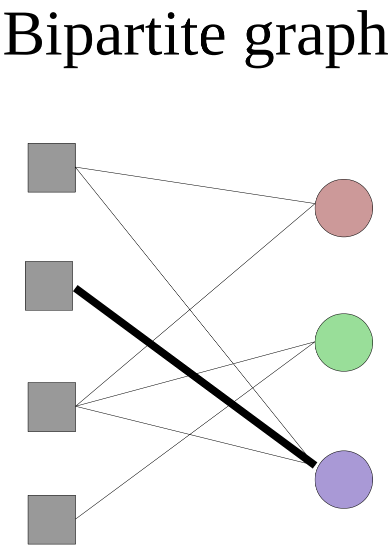


Bipartite graph



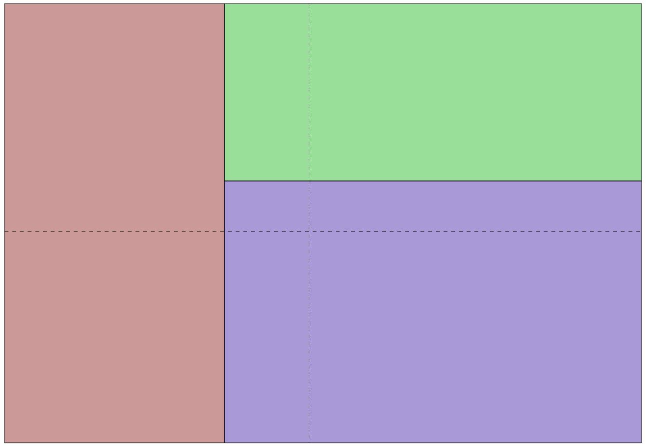
Bipartite graph matching



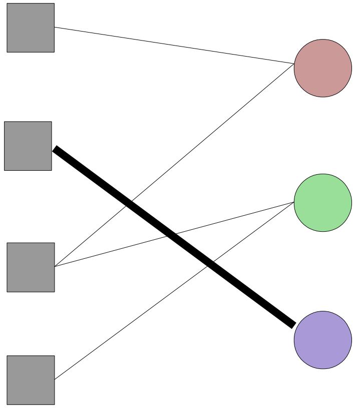


Bipartite graph matching

Partitioning

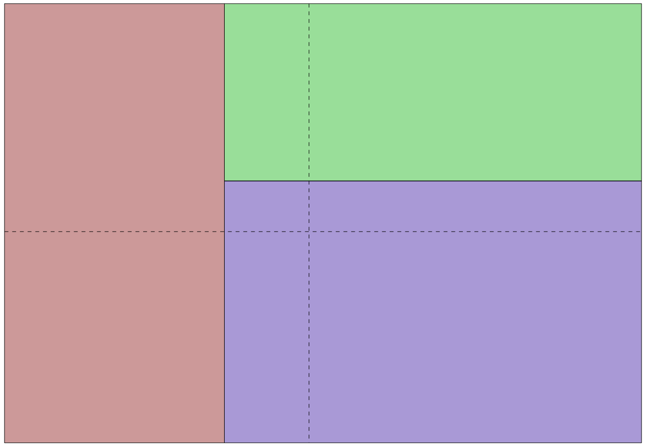


Bipartite graph

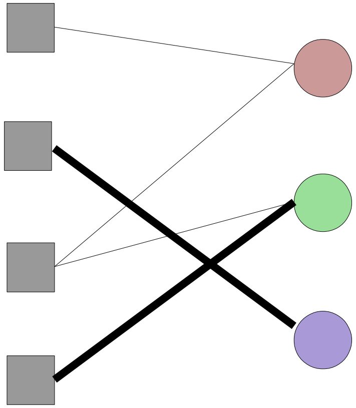


Bipartite graph matching

Partitioning

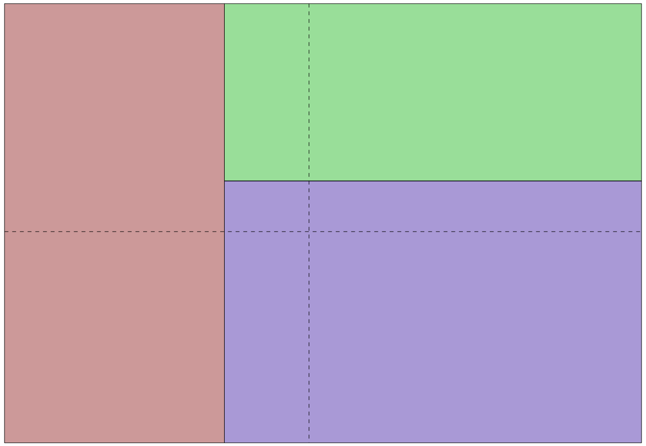


Bipartite graph

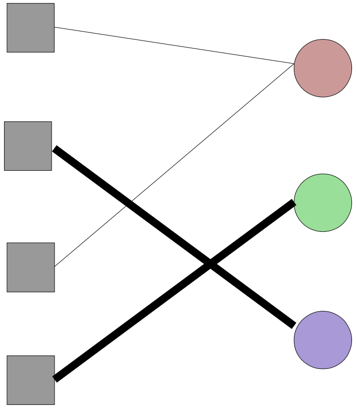


Bipartite graph matching

Partitioning

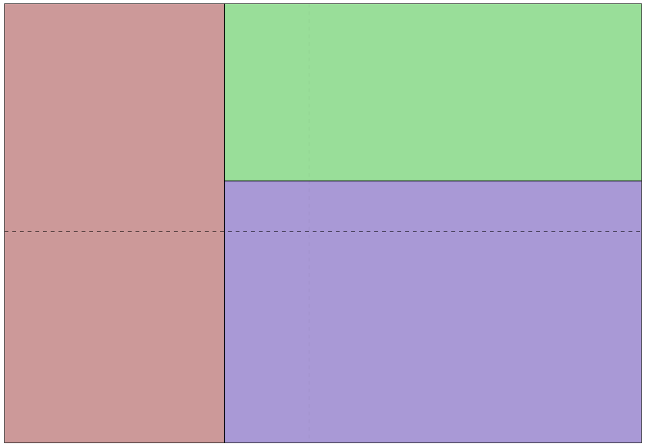


Bipartite graph

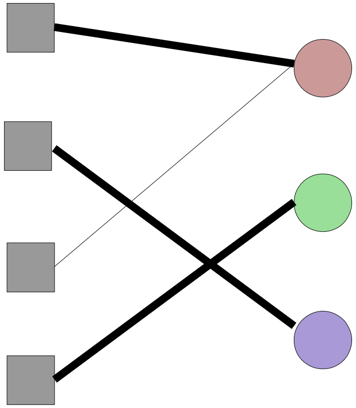


Bipartite graph matching

Partitioning

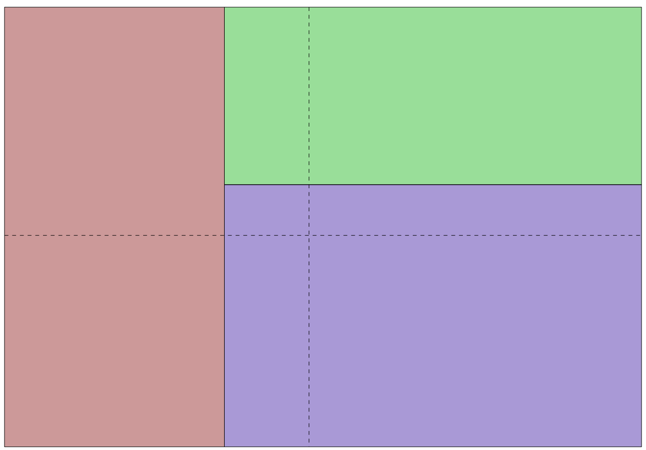


Bipartite graph

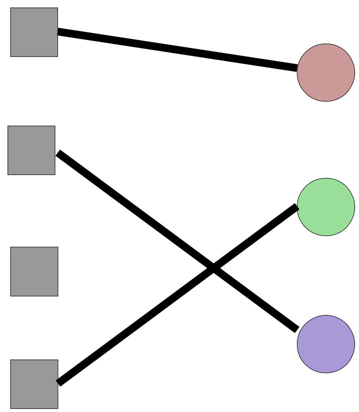


Bipartite graph matching

Partitioning

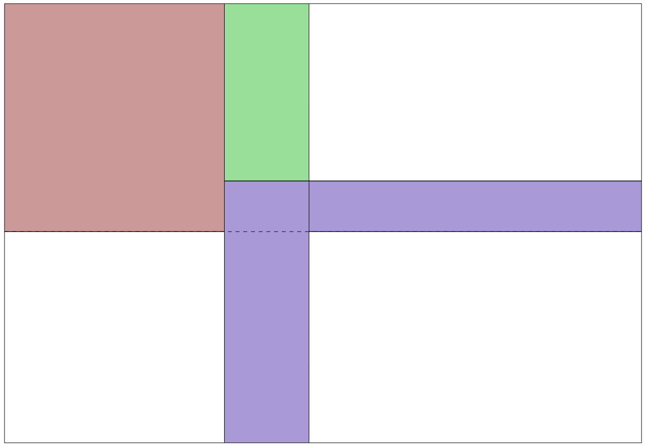


Bipartite graph



Bipartite graph matching

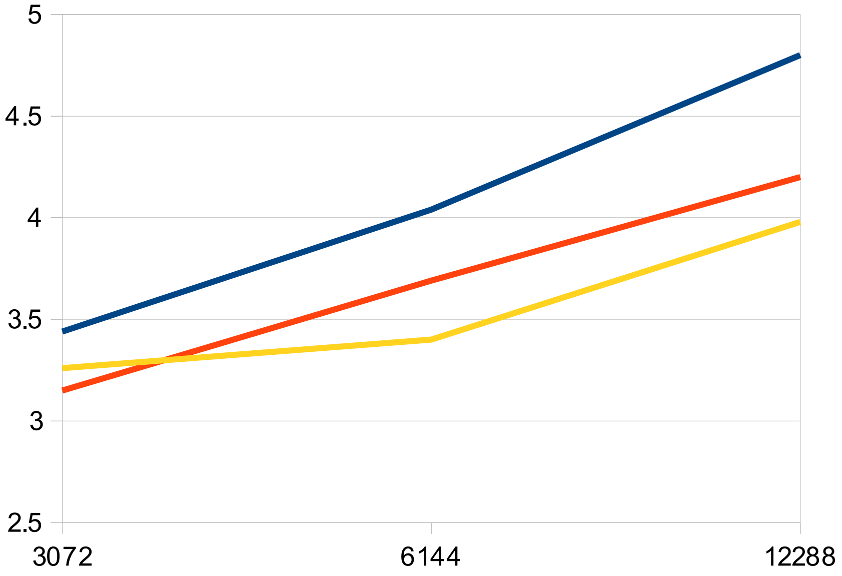
Partitioning



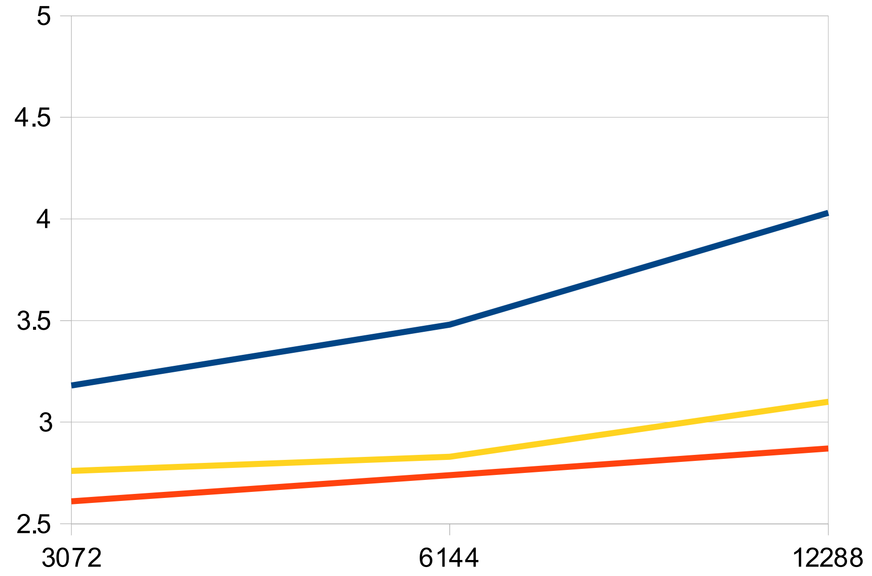
Results

Model problem: Laplace3D, 7-point stencil

Setup time



Solve time



none
task mapping
bipartite mapping

Conclusions / Future work

- Mapping of tasks to processors matters
- Reducing data migration seems to be more important than reducing solve kernels
- Careful mapping of data to processors may bring substantial benefits

- Combine two approaches for robustness
- Examine other mapping algorithms

Sandia National Laboratories